Name : VEDASHREE BHALERAO

Class : LY – AIA – 1

Batch : B

Roll No : 2213191

Assignment No. 3

Aim: Develop classification model for cat-dogs dataset using CNN model. Analyze the model accuracy and generate classification report.

- Analyze the result with and without regularization/dropout
- Develop an application and test the user given inputs **Objectives:**

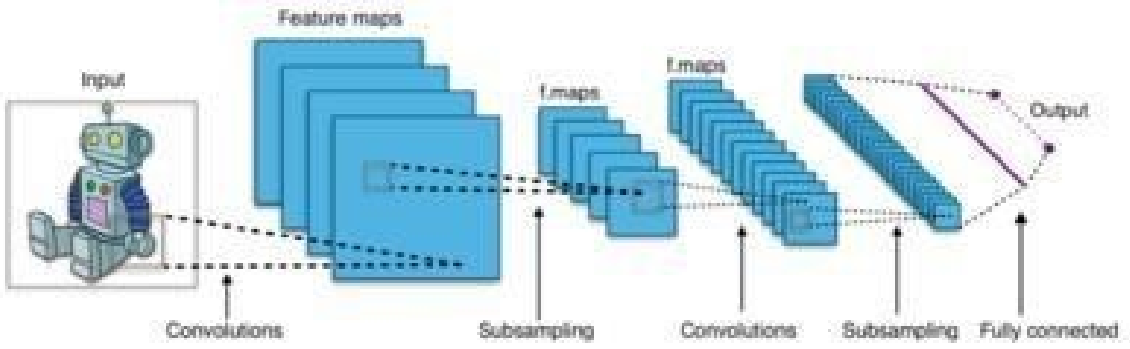
1. To learn about classification

2. To learn CNN

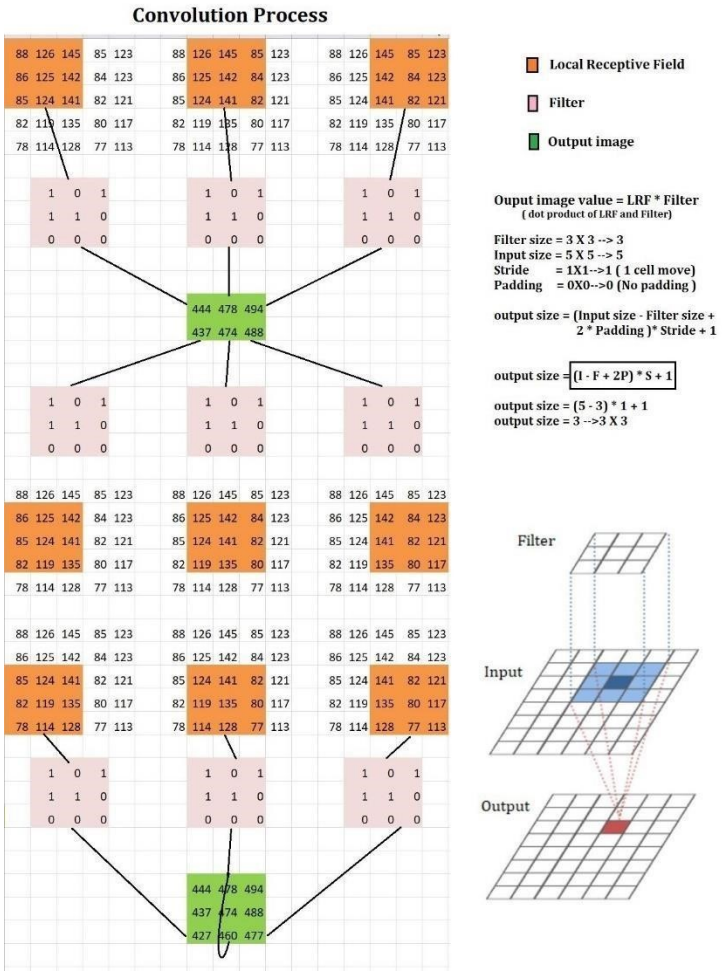
3. To demonstrate and analyse the results **Theory:**

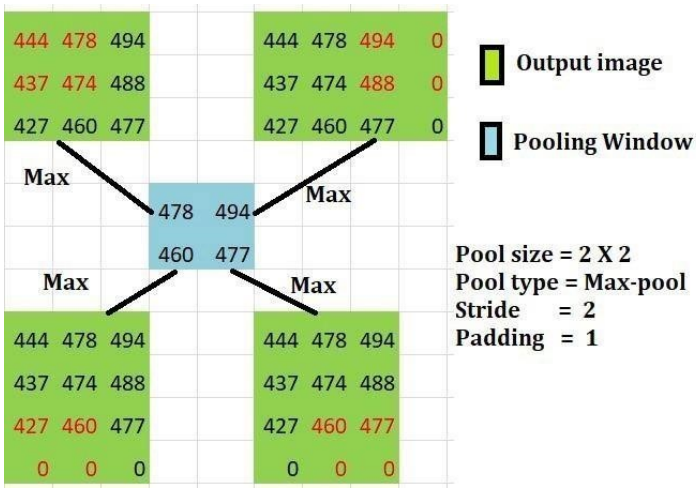
A **convolutional neural network** (**CNN**, or **ConvNet**) is a class of deep neural networks, most commonly applied to analysing visual imagery. They are also known as **shift invariant** or **space invariant artificial neural networks** (**SIANN**), based on their shared-weights architecture and translation invariance characteristics. They have applications in image and video recognition, recommender systems, image classification, medical image analysis, natural language processing, brain-computer interfaces, and financial time series.

CNNs are regularized versions of multilayer perceptron. Multilayer perceptron usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "fully-connectedness" of these networks makes them prone to overfitting data. Typical ways of regularization include adding some form of magnitude measurement of weights to the loss function. CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns. Therefore, on the scale of connectedness and complexity, CNNs are on the lower extreme.



Mathematics





Batch Normalization

Batch normalization is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-**batch**. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks.

Regularisation

This technique discourages learning a more complex or flexible model, so as to avoid the risk of overfitting. A simple relation for linear regression looks like this. Here Y represents the learned relation and β represents the coefficient estimates for different variables or predictors(X). $Y \approx \beta 0 + \beta 1X1 + \beta 2X2 + \ldots + \beta pXp$

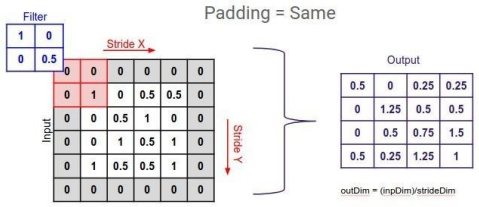
The fitting procedure involves a loss function, known as residual sum of squares or RSS. The coefficients are chosen, such that they minimize this loss function.

$$\mathrm{RSS} = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 .$$

Now, this will adjust the coefficients based on your training data. If there is noise in the training data, then the estimated coefficients won't generalize well to the future data. This is where regularization comes in and shrinks or regularizes these learned estimates towards zero.

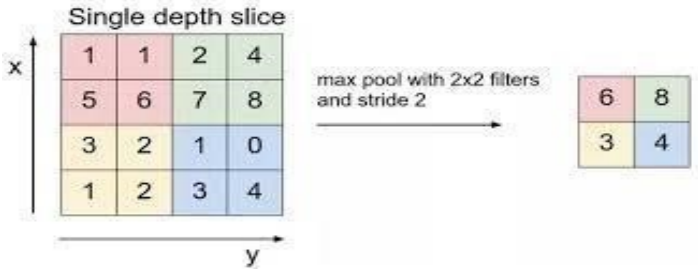
Padding

Padding is a term relevant to convolutional neural networks as it refers to the amount of pixels added to an image when it is being processed by the kernel of a **CNN**. For example, if the **padding** in a **CNN** is set to zero, then every pixel value that is added will be of value zero.



Strides

Stride is the number of pixels shifts over the input matrix. When the **stride** is 1 then we move the filters to 1 pixel at a time. When the **stride** is 2 then we move the filters to 2 pixels at a time and so on. The below figure shows convolution would work with a **stride** of 2.



Code:

```
 from tensorflow.keras.layers import Conv2D, MaxPooling2D from
tensorflow.keras.datasets import cifar10 from tensorflow.keras.utils
import to_categorical from tensorflow.keras.models import Sequential from
tensorflow.keras.layers import Dense, Flatten, Dropout from
tensorflow.keras.utils import to_categorical  # Add this import line
import numpy as np import tkinter as tk from tkinter import filedialog
from PIL import Image , ImageTk

# Load and preprocess CIFAR-10 data
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train = x_train.astype("float32") / 255.0 x_test =
x_test.astype("float32") / 255.0 y_train =
to_categorical(y_train, 10) y_test =
to_categorical(y_test, 10)

# Define the CIFAR-10 model cifar_model
= Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    MaxPooling2D((2, 2)),
    Dropout(0.2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.2),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(10, activation='softmax')
])  cifar_model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
cifar_model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=50,
batch_size=64)

cifar_labels = ["airplane", "automobile", "bird", "cat", "deer", "dog", "frog",
"horse", "ship", "truck"]

# GUI for uploading an image and testing the CIFAR-10 model def
upload_and_predict_cifar():
    file_path = filedialog.askopenfilename()
if file_path:
        # Load image and resize to 32x32         img =
Image.open(file_path).convert("RGB")  # CIFAR-10 expects RGB images
img_resized = img.resize((32, 32))
        # Display uploaded image on the screen
        img_display = ImageTk.PhotoImage(img_resized.resize((140, 140)))  #
Resize for better display in GUI
image_label.config(image=img_display)         image_label.image =
img_display
```

```
        # Normalize and reshape for model input         img_array =
np.array(img_resized).astype("float32") / 255.0         img_array =
img_array.reshape(1, 32, 32, 3)  # CIFAR-10 input shape
        # Make prediction and display result         prediction =
np.argmax(cifar_model.predict(img_array), axis=1)
result_label.config(text=f"Predicted Class:
{cifar_labels[prediction[0]]}")
```

GUI

```
# Tkinter GUI setup root = tk.Tk()
root.title("Image Recognition - CIFAR-10")

# GUI elements upload_button = tk.Button(root,
text="Upload Image",
command=upload_and_predict_cifar)
upload_button.pack()

# Label for displaying uploaded image
image_label = tk.Label(root) image_label.pack()

# Label for displaying prediction result result_label =
tk.Label(root, text="Prediction will appear here")
result_label.pack()

root.mainloop()
```

Results:

```
Epoch 1/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 21s 23ms/step - accuracy: 0.3208 - loss: 1.8374 - val_accuracy: 0.5454 - val_
loss: 1.2883
Epoch 2/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 18s 23ms/step - accuracy: 0.5241 - loss: 1.3276 - val_accuracy: 0.6004 - val_
loss: 1.1321
Epoch 3/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 18s 23ms/step - accuracy: 0.5827 - loss: 1.1748 - val_accuracy: 0.6395 - val_
loss: 1.0315
Epoch 4/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 17s 22ms/step - accuracy: 0.6164 - loss: 1.0863 - val_accuracy: 0.6675 - val_
loss: 0.9626
Epoch 5/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 19s 25ms/step - accuracy: 0.6410 - loss: 1.0142 - val_accuracy: 0.6690 - val_
loss: 0.9603
```

```
Epoch 46/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 22s 28ms/step - accuracy: 0.8193 - loss: 0.5043 - val_accuracy: 0.7445 - val_
loss: 0.8015
Epoch 47/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 19s 24ms/step - accuracy: 0.8168 - loss: 0.5019 - val_accuracy: 0.7414 - val_
loss: 0.8077
Epoch 48/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 22s 28ms/step - accuracy: 0.8151 - loss: 0.5134 - val_accuracy: 0.7436 - val_
loss: 0.8071
Epoch 49/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 20s 26ms/step - accuracy: 0.8216 - loss: 0.5000 - val_accuracy: 0.7385 - val_
loss: 0.8157
Epoch 50/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 20s 25ms/step - accuracy: 0.8230 - loss: 0.4872 - val_accuracy: 0.7448 - val_
loss: 0.8142
```

GUI

Conclusion:

Thus, we have understood how and where CNN is used and how it is programmed in TensorFlow.

We have also been able to polish GUI creation skills even further.